Lantian Qin

November 30, 2020

IT FDN 110 A

Assignment 07

<https://github.com/NoraQin/IntroToProg-Python-Mod07>

Pickling and Error Handling

Introduction

As we dive deeper into the world of programming, it’s useful to learn more about structured error handling and pickling in Python, as they are powerful tools that will help make our code more resilient and efficient. Pickling refers to serializing / de-serializing python object structures. Basically, it transforms a python object into a byte stream so that it can be saved in disk or sent over a network for later use. By storing the data in binary, it also reduces the file’s size. Structured error handling, on the other hand, is a completely different concept that involves using the try-except structure to handle errors, working with the exception objects, and creating custom exception objects for more powerful error handling. I will be sharing my knowledge on both topics separately in this document.

Review

By the end of the course, we are asked to think about the answer to the questions below. I will answer them to my best knowledge as a way to review the things I’ve learned.

* What are the benefits of putting built-in Python command into functions?
  + It could allow you to do different operations using one function call by specifying the parameters. This can make code cleaner when writing a program with menu options

* What are the benefits of using structured error handling?
  + It allows you to customize how your program handles errors instead of just letting Python do that for you. It is a good idea to add a try-except block to your programs whenever you think human interaction might cause a problem
* What are the differences between a text file and a binary file?
  + They are two different ways of storing data. Storing data in a binary format can obscure the file's

content and may reduce the file's size.

* How is the Exception class used?
  + It’s often used to hold information about an error Python automatically creates an Exception object when an error occurs. The Exception object automatically fills with information about the error that caused the exception.
* How do you "derive" a new class from the Exception class?
  + To do so, you create a new class derived from the exception class by running the following:

class myCustomExceptionClass(Exception): ……

* What is the markdown language?
  + Markdown language is commonly used to format simple texts for display. The markdown code is converted into a static HTML page using a conversion processing application called "Jekyll."
* How do you use Markdown on a GitHub webpage?
  + You can create a GitHub webpage and add formatted text, pictures, links and tables etc. to the webpage using Markdown.

Pickling

To use pickling with python objects, you should import the python module *pickle* which contains functions that will change a python object to binary format and vice versa.

Assuming you are trying to save a python object as a file to processing later or to send to someone else. The first thing you want to do is create a binary file in your local system that will hold the python object. To do that, you should use the open() function and set the keyword to *wb*. To help you remember, *w* stands for overwrite and *b* stands for binary.

In this example, I created a binary file for writing called *grocery\_list.txt*, and created a python list object called *grocery*. To load the list object into the binary file (also know as pickling), I used the *pickle.dump()* function, which takes the entire grocery object and essentially “dump” it into the binary file. After closing the binary file, I opened it in the local folder, and as you can see in Figure 2, it looks very messy. It’s not meant to be readable by human eyes since that in the processing of storing the object into a binary file, the object’s content has been obscured to reduce the file’s size.

Now let’s assume you received the binary file from someone and would like to load it into python for processing. In this case, you would first open the file for reading using the open() function. We use the *rb* keyword here as it stands reading a binary file.

With the file opened, use the pickle.load() function to un-pickle return the content of the file. As seen in Figure 3, when you run print(pickle.load(inlist)), Python console prints out the python list object you created earlier, so now you can save this into a value in Python and do data processing on it. Note that when you run the statement for the second time, Python throws an EOF error since pickle.load() reads one item in the binary file, or one “pickle” at a time. The list object we just loaded is the first and last pickle in the file, so Python can’t find anything to read. We will learn more about error handling in the next section.

If you are interested in learning more about pickling, I recommend checking out these websites:

<https://medium.com/@lokeshsharma596/what-is-pickle-in-python-3d9f261498b4> This explanation of pickling is pretty short and sweet, and the use cases are very helpful

<https://www.datacamp.com/community/tutorials/pickle-python-tutorial#what> This tutorial explains the concept of serialization pretty well

<https://stackoverflow.com/questions/20716812/saving-and-loading-multiple-objects-in-pickle-file> This Stack Overflow post helped me understand the concept of a “pickle” in a binary file, and how to read multiple pickles into Python efficiently

Graphical user interface, text, application, email

Description automatically generated

*Figure 1. Pickling demo 1*

Graphical user interface, text, application

Description automatically generated

*Figure 2. grocery\_list.txt*

Text

Description automatically generated

*Figure 3. python console output*

Error handling

**Try-except structure**

**We can use the try-except structure to make sure Python handles errors gracefully and asks it to do some custom operations in the case of error. As seen in the example in Figure 4, if you run print(1/0), python would throw a ZeroDivisionError. But if you put it in a try-except structure, and in the except block adds print(denominator is zero), then Python will first try to execute the code in the try block, and instead of throwing an error, Python would execute the code in the except block and print out “denominator is zero”.**

**Graphical user interface, text, application, email

Description automatically generated**

***Figure 4. Basic try-except structure***

**The Exception classes**

As you have probably noticed, Python can throw different types of errors depending on what the problem is. These errors are objects of the *Exception* class, which store information about specific kinds of error and are “raised” when the error occurs. Without error handling, they are automatically printed out when raised and ends the program, but you can use then in a try-except structure to print out more information about the error.

Each Exception object contains a number of methods, including \_\_class\_\_(), the name of the exception class, \_\_str\_\_(), a short description of the error, and \_\_doc\_\_(), a longer description of the error. Here in the example provided in Figure 5, I’ve deliberately created three different types of errors and printed out the details about each corresponding exception class. Notice that these three are all sub-classes of the master Exception class, therefore they are captured by the “*except Exception as e”* statement.

Graphical user interface, text, application

Description automatically generated

*Figure 5. The exception classes*

Graphical user interface, text, application, email

Description automatically generated

*Figure 6. Info about the exception classes*

**Capture specific exceptions**

Note that instead of using “except Exception as e” as a catch-all for all exceptions, you can also specify which kind of error you want to catch in the except-block. The examples in Figure 7 shows how it works. In the first portion, I created a zero division error in the try-block and set up the except-block so that it only catches zero division errors. As expected, Python executed the code in except-block and printed out information about the error. In the second portion however, the code in the try-block actually throws a file not found error. Since it’s not handled by the except-block, Python throws an error without executing what’s in the except-block. Note that you can actually use multiple except-blocks to capture and customize reaction to different types of errors, as seen in Figure 9.

Text

Description automatically generated

*Figure 7. Catching specific exceptions*

Text

Description automatically generated

*Figure 8. Catching specific exceptions – results*

Graphical user interface, text, application

Description automatically generated

*Figure 9. Use multiple except blocks*

**Custom Exception class**

Last but not least, you can created your own custom exception classes and raise them whenever you want. To do that, you want to create a custom class that inherits all the data and methods from the master Exception class by putting Exception in the parenthesis when declaring the class. In the class definition, you need to override the \_\_str\_\_() method to make it print out your custom message when the error is raised. You can also choose to override other methods from the Exception class such as \_\_doc\_\_(). In the example in Figure 10, I created a custom class named spaceInNameError, and raised it in the try-block in the next section of code. When I run the code and deliberately entered a file name with a space in the middle, Python prints out information about the Exception being raised, which is the spaceInNameError class I created. Note that if I just run the “raise spaceInNameError” statement without error handling, Python would print out an default error message just as it would for any other unhandled error.

Graphical user interface, text, application, email

Description automatically generated

*Figure 10. Custom Exception class*

Text

Description automatically generated

*Figure 11. Custom Exception class – result*

*Text

Description automatically generated*

*Figure 12. Raise a custom error on its own*

**Useful links**

<https://realpython.com/python-exceptions/> This is a great intro to Python Exceptions with tons of good examples

<https://docs.python.org/3/library/exceptions.html> This documentation contains a list of all the different types of Exceptions out there and what they mean

Summary

Pickling and error handling are quite useful for programmers in the real world. Pickling is commonly used in machine learning as it allows you to save your data to be used for prediction at a later time without having to read everything from txt file all over again or retrain the model. Error handling is extremely helpful in programs that deal with a lot of uncertainty (such as programs that collect user input or process data stream), as it can help provide users and developers with useful diagnostic information when errors occur.